## REMARKS

Claims 1, 13, 25, and 37-53 are pending in the application. Claims 1, 13, 25, and 37 are currently amended, while new claims 38-53 are added.

Claims 1, 13, 25, and 37 stand rejected under 35 U.S.C. §103(a) as unpatentable over U.S. Patent Application Publication No. 2004/0073541 to Lindblad et al., hereinafter, Lindblad, in view of "SAX2: Processing XML Efficiently with Java", O'Reilly Press, January 2002 authored by Brownell.

Applicants respectfully traverse the rejection based on the following discussion.

### I. The Prior Art Rejection under 35 U.S.C. §103(a) over Lindblad and Brownell

#### A. The Lindblad Reference

Lindblad discloses in Fig. 9 an XQuery server (XQE) 200 that includes a document processor 204 and a query processor 218. Document processor 204 generates step queries and step query results from [parsed] documents 202 and stores the step queries and step query results in a database 212. In one embodiment documents 202 are parsed documents. For example, parsed documents are created by an XML parsing process. The parsing process accepts XML textual inputs (serialized XML), analyzes the element structure of these documents, and outputs a data structure that represents the input document as a linked collection of element nodes linked to attribute nodes and child element nodes. The parsed XML document also may contain text nodes, processing instruction nodes, and comment nodes. (Paragraph **[0042]**).

Lindblad also discloses that an inverted file index may map terms to PostingLists. The terms correspond to textual units extracted from a collection of documents [parsed] 202 or document fragments from [parsed] documents 202, and PostingLists describe where and how often each term appeared within a given document or document fragment from [parsed] documents 202. (Paragraph **[0103]**, paragraph lines 2-7). Lindblad further discloses both subtree id's and scores may be kept in a differential form where each Posting stores only the encoded difference from the preceding subtree id and score. Large PostingLists typically have long strings of consecutive subtree id's with scores that are mostly equal. The PostingList

formats encode the consecutive runs using only one or two bits for the delta(id) (the id differential), and delta (score) (the score differential). Large PostingLists are stored with markers containing sufficient information to allow a search process to skip forward across blocks of Postings (a "skip-list" structure). The skip-list block size [is] a configurable parameter. (Paragraph **[0107]**, paragraph lines 3-14).

### B. The Brownell Reference

Brownell discloses that SAX is the API to use when you need to stream-process XML to conserve memory and, in most cases, CPU time. In SAX, handler interfaces call application (or library) code fro each significant chunk of XML information as it's parsed. These chunks include character data, elements, and attributes. Each event passes information to your code, which can save it or ignore as appropriate. These handlers see document information as a stream of such event calls, in "document order." Applications can process data incrementally, rather than in one big chunk, and they can discard information as soon as it's not needed. (Page 4, Stream-Based Processing, first paragraph).

Brownell also discloses that SAX parsers have several key advantages: Because SAX is a streaming API, it promotes pipelined processing, where I/O occurs while you use the CPU to do the work. You will naturally structure applications (or at least their SAX components) to use efficient single-pass algorithms and incremental processing. (Page 4, Stream-Based Processing, next to last paragraph).

Brownell further discloses that as soon as XML data starts to become available (perhaps over a network), SAX parsers start to provide it to applications. While processing element or character data, the network or the file system prefetches the next data. Such overlapped processing lowers latencies and makes good use of limited CPU cycles. With most other API's, your application won't even see data until the whole document has been fetched and parsed. (Page 4, Stream-Based Processing, last paragraph continued on page 5).

Brownell finally discloses that you can use SAX when your inputs aren't literal XML text. This is a powerful technique that helps you work with data at the level of parsed XML information (the XML "Infoset"). (Page 5, last bullet point).

### C. Argument

An aspect of the present invention discloses that there is a need for a novel technique that can reduce parsing time in the context of processing XQuery queries over XML documents. (Specification, Paragraph **[0005]**, page 5, lines 1 and 2). The inventor also discloses in the Description of Related Art that "[m]ost of the XPath and XQuery implementations today process queries by traversing an in-memory representation of the document using the Document Object Module (DOM) interface. In DOM, at any point, the processing can move in any direction in the XML tree from the current node to its children, its parent or any of its siblings. While this makes the implementation easier, the requirement that the whole document be saved in memory is a major drawback of this approach, leading to large memory consumption (decreased concurrency) and high latency (the document needs to be processed before the first answer is produced)." (Specification, Paragraph **[0003]**).

Claims 1 and 25 recite in relevant part,

"producing, by said streaming API for a mark-up language data stream, an ordered index of all textual elements corresponding to their order in said mark-up language data stream, said ordered index comprising tag identifiers and end positions corresponding to each of said all textual elements".

Similarly, claims 13 and 37 recite in relevant part,

"an ordered index of all textual elements corresponding to their order in said mark-up language data stream, said ordered index comprising tag identifiers and end positions corresponding to each of said all textual elements and being produced by said streaming API for a mark-up language data stream".

Nowhere does Lindblad disclose, teach or suggest the present invention's feature of "producing, by said streaming API for a mark-up language data stream, an ordered index of all textual elements corresponding to their order in said mark-up language data stream, said ordered index comprising tag identifiers and end positions corresponding to each of said all textual elements."

Instead, Lindblad discloses a method of parent-child query indexing for XML databases using conventional XQuery processing. That is, Lindblad processes the entire XML document which includes a step query generator 206, a canonicalizer 208, and a hash key generator 210, before processing by a step query result generator. (Paragraph **[0046]**, paragraph lines 2-5). Lindblad cannot process a mark-up language data stream as does the present invention.

Nowhere does Lindblad disclose, teach or suggest using a streaming API for a mark-up language data stream. In fact, Lindblad uses the conventional XQuery document processing for an entire processed document, which the present invention avoids by use of the ordered index produced by the streaming API for a mark-up language data stream, which is then accessed by the query processor.

Furthermore, Linblad's invocation of skipping, which the Office Action analogizes to the present invention's feature of "skipping an unmatched textual element for parsing, if a tag identifier, corresponding to said unmatched textual element, does not match said query", does not skip unmatched (that is, not matched by a query processor to tag identifiers stored in the ordered index) textual elements for parsing. Instead, Lindblad skips forward across blocks of Postings, a skip-list, where the blocks differ insignificantly in score or subtree id. The skip blocks of Lindblad have already been parsed! Hence, there is no reduction in parsing to a query by Lindblad.

For at least the reasons outlined above, Applicants respectfully submit that Lindblad does not disclose, teach or suggest the present invention's claimed feature of "producing, by said streaming API for a mark-up language data stream, an ordered index of all textual elements corresponding to their order in said mark-up language data stream, said ordered index comprising tag identifiers and end positions corresponding to each of said all textual elements".

Although the SAX API disclosed by Brownell may be used to produce the ordered index of the present invention, nowhere does Brownell disclose, teach or suggest the present invention's feature of "producing, by said streaming API for a mark-up language data stream, an ordered index of all textual elements corresponding to their order in said mark-up language data stream, said ordered index comprising tag identifiers and end positions corresponding to each of

said all textual elements." (emphasis added).

Instead, the index disclosed by Brownell at page 191, lines 10-12 (cited by the Office Action at page 5, lines 5 and 6),

"public String getName (int index);
public String getType (int index);
public String getValue (int index);" could more properly be described as a pointer to strings for getName, getType, and getValue, respectively.

For at least the reasons outlined immediately above, and for those reasons outlined above with respect to the rejection over Lindblad, Applicants respectfully submit that Lindblad and Brownell, either individually or in combination, do not disclose, teach or suggest the present invention's claimed feature of "producing, by said streaming API for a mark-up language data stream, an ordered index of all textual elements corresponding to their order in said mark-up language data stream, said ordered index comprising tag identifiers and end positions corresponding to each of said all textual elements". Accordingly, Lindblad and Brownell, either individually or in combination, do not render obvious the subject matter of currently amended independent claims 1, 13, 25, and 37, and new dependent claims 38-53 under 35 U.S.C. §103(a). Withdrawal of the rejection of claims 1, 13, 25, and 37 under 35 U.S.C. §103(a) is respectfully solicited.

## II. Formal Matters and Conclusion

Claims 1, 13, 25, and 37-53 are pending in the application.

With respect to the prior art rejection of the claims, the claims are amended, above, to overcome the rejection. In view of the foregoing, the Examiner is respectfully requested to reconsider and withdraw the rejections to the claims.

Applicants respectfully submit that claims 1, 13, 25, and 37-53, all the claims presently pending in the application, are patentably distinct from the prior art of record and are in condition for allowance. The Examiner is respectfully requested to pass the above application to issue at the earliest possible time.

Should the Examiner find the application to be other than in condition for allowance, the

Examiner is requested to contact the undersigned at the local telephone number listed below to discuss any other changes deemed necessary. Please charge any deficiencies and credit any overpayments to Attorney's Deposit Account Number 09-0441.

Respectfully submitted,

Dated: December 6, 2007

/Peter A. Balnave/
Peter A. Balnave, Ph.D.
Registration No. 46,199

Gibb & Rahman, LLC
2568-A Riva Road, Suite 304
Annapolis, MD 21401
Voice: (410) 573-5255
Fax: (301) 261-8825
Customer Number: 29154